



Criação de API para Banco de Dados

API Restfull

O REST é uma especificação técnica sobre intercomunicação heterogênea para aplicações Web.

A adoção do REST pode levar a uma arquitetura simples, escalável, eficaz, segura e confiável. REST é um protocolo RPC leve construído sobre o protocolo HTTP, sua simplicidade e facilidade na web o torna um alternativa inigualável ao SOAP, uma solução RPC popular há anos. Uma aplicação Web bem estruturada pode ser facilmente construída ao usar o Rest, muitos desenvolvedores criaram com sucesso uma base de API simples e robusta no serviço web Ajax e Restful.

A transferência de estado representacional (REST) ou serviços Web RESTful são uma maneira de fornecer interoperabilidade entre sistemas de computador na Internet. Os serviços da Web compatíveis com REST permitem que os sistemas solicitantes acessem e manipulem representações textuais de recursos da Web usando um conjunto uniforme e predefinido de stateless operations .

O termo transferência de estado representacional foi introduzido e definido em 2000 por Roy Fielding em sua tese de doutorado. Fielding usou REST para projetar HTTP e Uniform Resource Identifiers (URI). Um recurso é um tipo de informação que pode ser acessada, como um aplicativo objeto, um registro de banco de dados, um algoritmo e assim por diante. Cada recurso é identificado

por um URI exclusivo (Universal Resource Identifier), REST representam URI na forma de "/user/name", e operações em métodos HTTP GET, PUT, POST, DELETE, HEADER e OPTIONS, resultando no próximo recurso sendo transferido de volta para o chamador. Uma característica importante do REST é que o lado do servidor mantém sem estado entre várias interações, cada servidor nos clusters pode atender o cliente em cada solicitação.

E quando falamos dos endpoints de APIs Rest Restfuul, ela executa uma operação usando 'solicitações' e 'respostas'. Se as APIs enviarem uma informação de 'solicitação' de um aplicativo ou servidor Web, ele receberá uma 'resposta'. Com APIs REST, um endpoint é uma extremidade de um canal de comunicação. Cada endpoint é um local onde as APIs REST podem acessar os recursos necessários para realizar uma função.

E falando no futuro da API RESTful REST, de acordo com as estatísticas, diz-se que 82% das APIs eram RESTful que são OpenAPI ou Swagger e 21% eram RESTful sem OpenAPI. Há apenas uma pequena queda no uso de APIs REST em APIs públicas. O GraphQL é um concorrente próximo das APIs REST RESTful. Essas APIs sempre permanecerão supremas e não serão afetadas diretamente por outros concorrentes do mercado.

Métodos PUT, POST, DELETE e UPDADE

Os verbos HTTP compreendem a maior parte de nossa restrição de "interface uniforme" e nos fornecem a contrapartida de ação para o recurso baseado em substantivos. Os verbos HTTP primários ou mais usados (ou métodos, como são chamados corretamente) são POST, GET, PUT e DELETE. Correspondem a operações de criação, leitura, atualização e exclusão (ou CRUD), respectivamente. Existem

vários outros verbos também, mas são utilizados com menos frequência. Desses métodos menos frequentes, OPTIONS e HEAD são usados com mais frequência do que outros.

Abaixo está uma tabela que resume os valores de retorno recomendados dos métodos HTTP primários em combinação com os URIs de recursos:

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

Tabela 1 - Valores de retorno métodos HTTP

Método GET

O método HTTP GET é usado para **ler** (ou recuperar) uma representação de um recurso. No caminho "happy" (ou sem erro), GET retorna uma representação em XML ou JSON e um código de resposta HTTP de 200 (OK). Em um caso de erro, na maioria das vezes ele retorna um 404 (NOT FOUND) ou 400 (BAD REQUEST).

De acordo com o design da especificação HTTP, as solicitações GET (juntamente com HEAD) são usadas apenas para ler dados e não alterá-los. Portanto, quando usados dessa forma, são considerados

seguros. Ou seja, eles podem ser chamados sem risco de modificação ou corrupção de dados - chamá-los uma vez tem o mesmo efeito que chamá-los 10 vezes, ou nenhum. Além disso, GET (e HEAD) é idempotente, o que significa que fazer várias requisições idênticas acaba tendo o mesmo resultado de uma única requisição.

Não exponha operações inseguras via GET — ele nunca deve modificar nenhum recurso no servidor.

Método POST

O verbo POST é usado com mais frequência para **criar** novos recursos. Em particular, é usado para criar recursos subordinados. Ou seja, subordinado a algum outro recurso (por exemplo, pai). Em outras palavras, ao criar um novo recurso, POST passa o pai e o serviço se encarrega de associar o novo recurso ao pai, atribuir um ID (novo URI de recurso), etc.

Na criação bem-sucedida, retorne o status HTTP 201, retornando um cabeçalho Location com um link para o recurso recém-criado com o status HTTP 201.

Fazer duas solicitações POST idênticas provavelmente resultará em dois recursos contendo as mesmas informações.

Método PUT

O método PUT é usado com mais frequência para recursos de **atualização**, PUT para um URI de recurso conhecido com o corpo da solicitação contendo a representação recém-atualizada do recurso original.

No entanto, PUT também pode ser usado para criar um recurso no caso em que o ID do recurso é escolhido pelo cliente em vez do servidor. Em outras palavras, se o PUT for para uma URI que contenha o valor de um ID de recurso inexistente. Novamente, o corpo da solicitação contém uma representação de recurso. Muitos acham que isso é complicado e confuso. Consequentemente, este método de criação deve ser usado com moderação, se for o caso.

Como alternativa, use POST para criar novos recursos e fornecer o ID definido pelo cliente na representação do corpo — presumivelmente para uma URI que não inclui o ID do recurso.

Na atualização bem-sucedida, retorne 200 (ou 204 se não estiver retornando nenhum conteúdo no corpo) de um PUT. Se estiver usando PUT para criar, retorne o status HTTP 201 na criação bem-sucedida. Uma resposta no body é opcional, desde que consuma mais largura de banda. Não é necessário retornar um link por meio de um cabeçalho Location no caso de criação, pois o cliente já definiu o ID do recurso.

PUT não é uma operação segura, pois modifica (ou cria) o estado no servidor, mas é idempotente. Em outras palavras, se você criar ou atualizar um recurso usando PUT e fizer a mesma chamada novamente, o recurso ainda estará lá e ainda terá o mesmo estado da primeira chamada.

Método DELETE

O método DELETE é muito fácil de entender. Ele é usado para **excluir** um recurso identificado por um URI.

Na exclusão bem-sucedida, retorne o status HTTP 200 (OK) junto com um corpo de resposta, talvez a representação do item excluído

(geralmente exige muita largura de banda) ou uma resposta agrupada. Isso ou retornar o status HTTP 204 (SEM CON DO) sem corpo de resposta. Em outras palavras, um status 204 sem corpo ou a resposta no estilo JSEND e o status HTTP 200 são as respostas recomendadas.

Em termos de especificação HTTP, as operações DELETE são independentes. Se você EXCLUIR um recurso, ele será removido. Chamar repetidamente DELETE nesse recurso acaba da mesma forma: o recurso se foi. Se chamar DELETE digamos, decrementa um contador (dentro do recurso), a chamada DELETE não é mais válida. Conforme mencionado anteriormente, as estatísticas e medições de uso podem ser atualizadas enquanto ainda se considera o serviço adequado, desde que nenhum dado do recurso seja alterado. Recomenda-se o uso de POST para solicitações de recursos não dependentes.

Há uma ressalva sobre o DELETE, no entanto. Chamar DELETE em um recurso uma segunda vez geralmente retornará um 404 (NOT FOUND), pois ele já foi removido e, portanto, não é mais localizável. Isso, segundo algumas opiniões, faz com que as operações DELETE não sejam mais relevantes, no entanto, o estado final do recurso é o mesmo. Retornar um 404 é aceitável e comunica com precisão o status da chamada.

Implementando uma API com Ionic

Ionic é um SDK de código aberto completo para desenvolvimento de aplicativos móveis híbridos criado por Max Lynch, Ben Sperry e Adam Bradley da Drifty Co. em 2013. Sua versão original se deu no ano de 2013 e construída sobre AngularJS e Apache Cordova. No entanto, a versão mais recente foi reconstruída como um conjunto de Web

Components, permitindo que o usuário escolha qualquer estrutura de interface do usuário, como Angular, React ou Vue.js. Ele também permite o uso de componentes Ionic sem nenhuma estrutura de interface de usuário. A Ionic fornece ferramentas e serviços para o desenvolvimento de aplicativos híbridos móveis, desktop e web progressivos com base em tecnologias e práticas modernas de desenvolvimento web, usando tecnologias web como CSS, HTML5 e Sass. Em particular, os aplicativos móveis podem ser construídos com essas tecnologias da Web e depois distribuídos por meio de lojas de aplicativos nativos para serem instalados em dispositivos utilizando Cordova ou Capacitor.

Os benefícios dos aplicativos híbridos são os seguintes:

- Você só tem um código base.
- Você pode desenvolver para várias plataformas ao mesmo tempo.
- Velocidade no desenvolvimento.
- Os sensores e componentes nativos dos dispositivos podem ser usados.

Antes de começar a consumir uma API REST, você precisa atualizar ou instalar alguns serviços na máquina.

1. Instale o NodeJS (https://nodejs.org/en/).
2. Córdoba (sudo npm install -g cordova).
3. CLI Iônico (sudo npm install -g ionic).

Uma vez instalado, podemos começar a criar nosso aplicativo.

Agora vamos criar nosso primeiro aplicativo lembrando que ionic funciona multiplataforma Web, Android e iOS.

ionic start pratica-integradora tabs --type=angular –capacitor

Agora fazendo uma analogia a REST API e o protocolo HTTP com métodos POST,GET, PUT e DELETE

Nesse momento vamos criar um arquivo de service na pasta services, posteriormente vamos adicionar novos services para outros objetos.

ionic g service services/gas

Agora vamos dar início a construção do arquivo service.

```
export class GasService {

 BASE_URL: string = 'http://localhost:3000/';

 constructor(private http: HttpClient) { }
```

Em nosso gas service vamos adicionar as funções conforme os protocolos.

```
 getGasStation(): Observable<GasStation[]> {

  var url: string = this.BASE_URL + 'gas-station';

  return this.http.get<GasStation[]>(url)

 }
```

A função acima retorna todos os postos cadastrados.



```
addGasStation(gas: any): Observable<GasStation> {

  console.log(user);

  var url: string = this.BASE_URL + 'gas-station';

  return this.http.post<GasStation>(url, gas, httpOptions);

 }
```

A função acima salva o objeto GasStation enviando o mesmo por post através da transformação do objeto em Json.

```
 editGasStation(gas: any): Observable<GasStation> {

  var url: string = this.BASE_URL + 'gas-station/' + gas.id;;

  return this.http.put<GasStation>(url, gas, httpOptions);

 }

 deleteGasStation(gas: GasStation | string): Observable<GasStation>
{

  const id = typeof user === 'string' ? gas : gas.id;

  var url: string = this.BASE_URL + 'gas-station/' + id;

  return this.http.delete<GasStation>(url, httpOptions);

 }
```

As duas funções são as últimas de nosso service a primeira edita e segunda apaga os dados utilizando os métodos put e delete.

Podemos agora comparar o arquivo gas.service com a fig 1 e fazer uma analogia ao métodos HTTP e o CRUD.

```
import { Injectable } from '@angular/core';

import { HttpClient, HttpHeaders } from '@angular/common/http';

import { Observable } from 'rxjs';

import { GasStation } from '../models/gas-station';

const httpOptions = {

 headers: new HttpHeaders({ 'Content-Type': 'application/json' })

};

@Injectable({

 providedIn: 'root'

})

export class GasService {

 BASE_URL: string = 'http://localhost:3000/';

 constructor(private http: HttpClient) { }

 / GET gas api  */

 getGasStation(): Observable<GasStation[]> {

  var url: string = this.BASE_URL + 'gas-station';
```

```
return this.http.get<GasStation[]>(url)
}

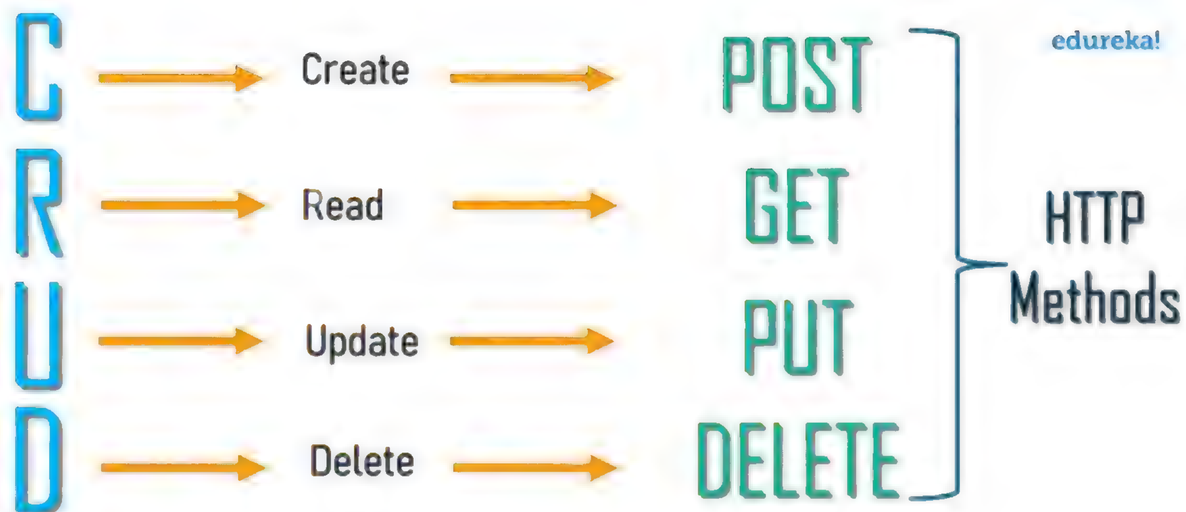
addGasStation(gas: any): Observable<GasStation> {
  console.log(gas);
  var url: string = this.BASE_URL + 'gas-station';
  return this.http.post<GasStation>(url, gas, httpOptions);
}
```



```
/* PUT gas api EDIT gas Function */
editGasStation(gas: any): Observable<GasStation> {
  var url: string = this.BASE_URL + 'gas-station/' + gas.id;
  return this.http.put<GasStation>(url, gas, httpOptions)
}

deleteGasStation(gas: GasStation | string): Observable<GasStation>
{
  const id = typeof gas === 'string' ? gas : gas.id
  var url: string = this.BASE_URL + 'gas-station/' + id
  return this.http.delete<GasStation>(url, httpOptions)
}
```





Posteriormente, durante nosso projeto de posto de gasolina vamos utilizar os services para comunicar com uma API e criar os respectivos cadastros necessários para nossa aplicação.

Ferramentas para consumir API

O teste de consumo da API consiste em um software que testa como as APIs foram desenvolvidas. Ele analisa se as APIs cumprem corretamente sua finalidade para garantir que a funcionalidade, confiabilidade, segurança e desempenho do aplicativo não sejam comprometidos. Além disso, o consumo de API envolve principalmente os testes de serviços Web SOAP ou APIs REST, usando cargas úteis de mensagens JSON ou XML, que são enviadas por meio de HTTPS, HTTP, MQ e JMS.

As ferramentas de teste de API permitem que os testadores verifiquem vários aspectos, incluindo:

- se uma API estiver retornando a resposta espera[illegible] no formato correto;
- se reage adequadamente a casos extremos (por exemplo, falhas e entradas inesperadas);
- se responde com segurança a possíveis ataques de segurança;
- O tempo que leva para entregar uma resposta.

Podemos destacar algumas ferramentas para fazer o consumo e testes das APIs, como por exemplo:

Postman

É um ambiente de desenvolvimento de API. O Postman API Development Environment é dividido em três partes, Collections, Workspaces e Built-in Tools. As coleções do Postman permitem executar solicitações, testar e depurar, criar testes automatizados e simular, documentar e monitorar a API.

O espaço de trabalho do Postman fornecerá os recursos de colaboração. Ele permitirá que você compartilhe as coleções, defina permissões e gerencie a participação em vários espaços de trabalho para qualquer tamanho de equipe. As ferramentas integradas fornecerão os recursos que serão exigidos pelos desenvolvedores para trabalhar com uma API.

Recursos:

- Ajuda em testes automatizados.

- Auxilia em testes exploratórios.



- Ele suporta os formatos Swagger e RAML (RESTful API Modeling Language).

- Ele suporta o compartilhamento de conhecimento dentro da equipe.

Insomnia

Um cliente REST é uma ferramenta usada para interagir com uma API RESTful que é exposta para comunicação. Um cliente REST Insomnia é um cliente de API REST poderoso e de código aberto usado para armazenar, organizar e executar solicitações de API REST de forma elegante. O Insomnia REST Client é uma excelente alternativa ao Postman para envio de solicitações REST e GraphQL com suporte para gerenciamento de cookies, variáveis de ambiente, geração de código e autenticação. Está disponível para os sistemas operacionais Windows, Mac e Linux. Além disso, o Insomnia incorpora uma GUI amigável com recursos sofisticados, como auxiliares de segurança, criação de código e variáveis de ambiente.

A seguir estão alguns dos recursos do Insomnia REST Client:

- Suporte multiplataforma;

- Capacidade de executar solicitações REST, SOAP, GraphQL e GRPC;

- Capacidade de armazenar, organizar e executar solicitações de API REST;

- Capacidade de organizar solicitações em espaços de trabalho e grupos;
- Suporte para construtor de parâmetros de string de consulta;
- Capacidade de exportar e compartilhar espaços de trabalho;
- Suporte para solicitações encadeadas.

Atividade Extra

Para se aprofundar no assunto desta aula assista o vídeo de Como Consumir Web Service Api direto no SQL Server e Mapear JSON do professor Drausio.

Canal:

Título a ser buscado no youtube: professordrausio

Link do trailer: **Como Consumir Web Service Api direto no SQL Server e Mapear JSON**

Referência Bibliográfica

DRIFTY, Inc (2016). **"Ionic Documentation Overview - License"**.

FIELDING, Roy Thomas (2000). "**Chapter 5: Representation ⓘ tate Transfer (REST)".**

PEREZ, Sarah. **"Drifty, Makers Of The Ionic Mobile Framework, Raise $1 Million"**. Visto em 11/05/2022.

ZHOU, Qiao Jun. **The Design and Implementation of RESTful Web Services Open Platform[D].**Zhejiang University,2016.

"Web Services Architecture". World Wide Web Consortium. 11 February 2004. 3.1.3 Relationship to the World Wide Web and REST Architectures. Visto em 11/05/2022.

Atividade Prática 9 - Integrar Banco de Dados via API

Título da Prática: Criação de API para Banco de Dados

Objetivos: Utilizar o insomnia para fazer os testes com uma API para banco de dados.

Materiais, Métodos e Ferramentas: Iremos fazer uma pesquisa na internet e utilizar a ferramenta Insomnia para criar os testes com a API.

Atividade Prática

Insomnia

Um cliente REST é uma ferramenta usada para interagir com uma API RESTful que é exposta para comunicação. Um cliente REST Insomnia é um cliente de API REST poderoso e de código aberto usado para

armazenar, organizar e executar solicitações de API REST de forma elegante. O Insomnia REST Client é uma excelente alternativa ao Postman para envio de solicitações REST e GraphQL com suporte para gerenciamento de cookies, variáveis de ambiente, geração de código e autenticação. Está disponível para os sistemas operacionais Windows, Mac e Linux. Além disso, o Insomnia incorpora uma GUI amigável com recursos sofisticados, como auxiliares de segurança, criação de código e variáveis de ambiente.

A seguir estão alguns dos recursos do Insomnia REST Client:

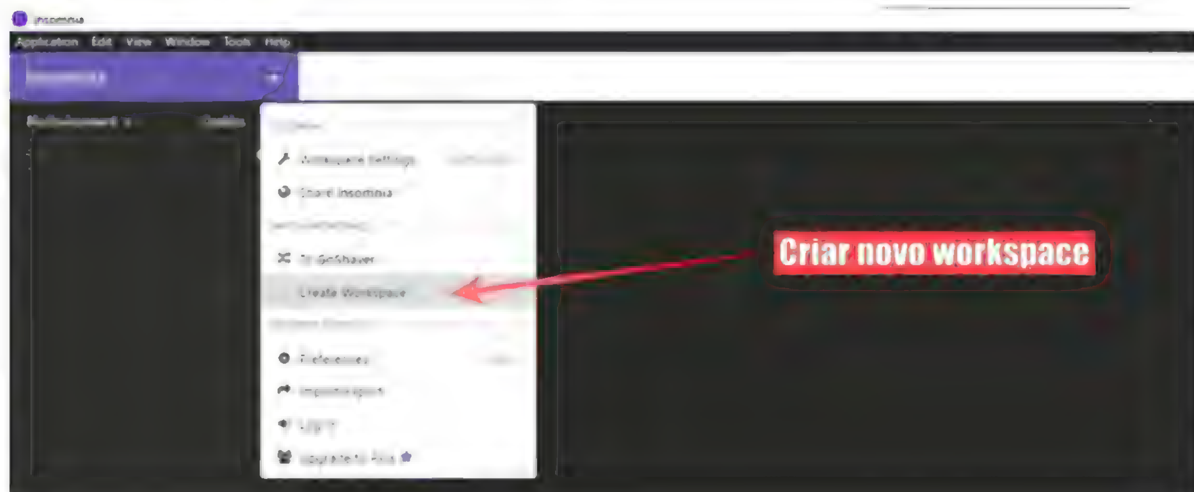
- Suporte multiplataforma.
- Capacidade de executar solicitações REST, SOAP, GraphQL e GRPC.
- Capacidade de armazenar, organizar e executar solicitações de API REST.
- Capacidade de organizar solicitações em espaços de trabalho e grupos.
- Suporte para construtor de parâmetros de string de consulta.
- Capacidade de exportar e compartilhar espaços de trabalho.
- Suporte para solicitações encadeadas.

A proposta desta atividade prática é criar alguns testes de API através do insomnia. Por exemplo, caso não tenha uma API pronta

para testar pode utilizar de algumas APIs públicas, como é o caso da SWAPI (https://swapi.dev/).

Colocamos abaixo um passo a passo para fazer um teste com o insomnia:

Primeiramente vamos criar um Workspace para a API, como segue na figura abaixo:



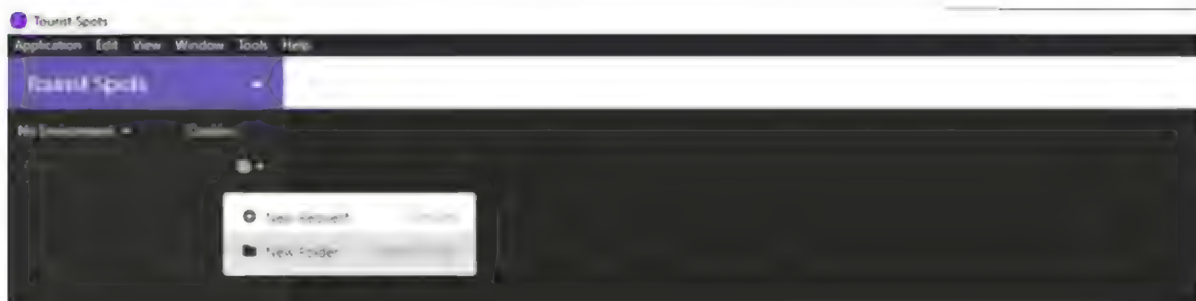
Vamos colocar um nome de sua preferência e finalizar. Após a criação do workspace irá aparecer um botão azul do lado esquerda que indicará qual estará em uso no momento:



E para organizar melhor nossas informações vamos criar algumas pastas. Mesmo que no seu caso esteja trabalhando com requisições

somente de uma aplicação, é bem comum haver um número variado de entidades diferentes na mesma aplicação. Por exemplo, nesta aplicação que estamos mostrando, teremos requisições para pontos turísticos, para algumas atrações, para situações de avaliações, para comentários e outros... As pastas, nesse cenário, irão nos ajudar a organizar as nossas requisições pela natureza de cada uma.

A primeira requisição, que iremos mostrar mais adiante, será uma requisição que iremos receber todas as atrações turísticas. Vamos criar, então, a pasta *Atrações*. Para isso, basta acionar o atalho *Shift + Ctrl + N* ou se preferir ir à opção *New Folder*, conforme imagem a seguir:

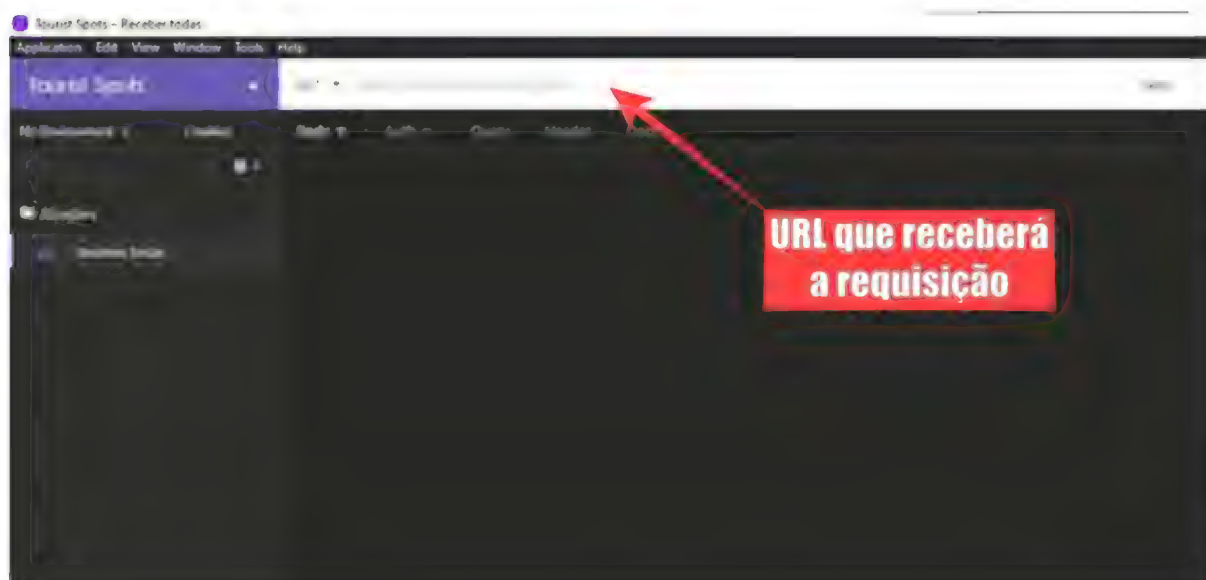


Assim daremos um nome para a pasta, e na sequência você irá perceber que a sua pasta criada irá aparecer no menu esquerdo.

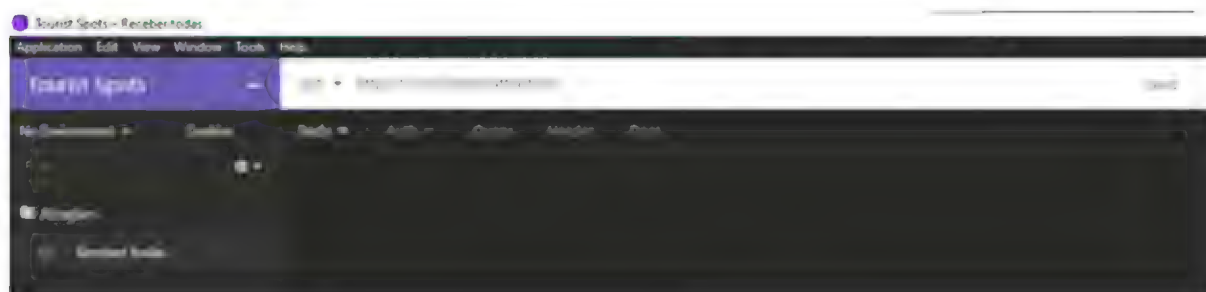
Para fazer uma primeira requisição vamos na opção *New Request* e vamos criar uma requisição do tipo GET.



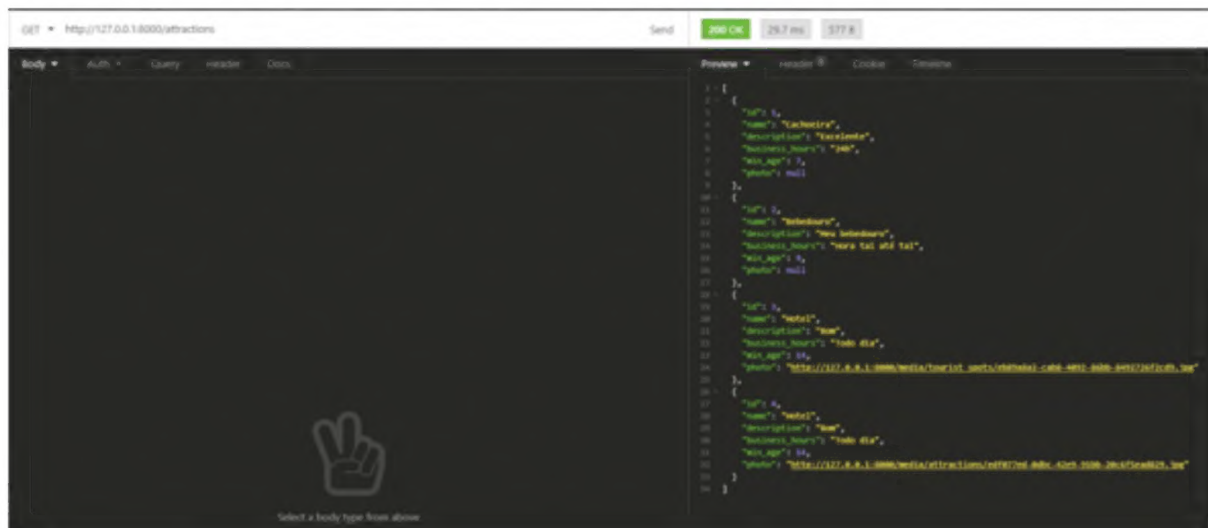
Você pode colocar um nome para a nova requisição e definir qual método HTTP que essa sua requisição vai usar. Em seguida, confirme que deseja criá-la. É interessante perceber que, após nós adicionarmos a primeira requisição, a interface inicial vai sofrer uma alteração. Agora, a nova requisição que acabou de criar irá aparecer no menu à esquerda. O que precisamos fazer, em seguida, é selecionar essa requisição e assim identificar o local em que colocaremos sua URL:



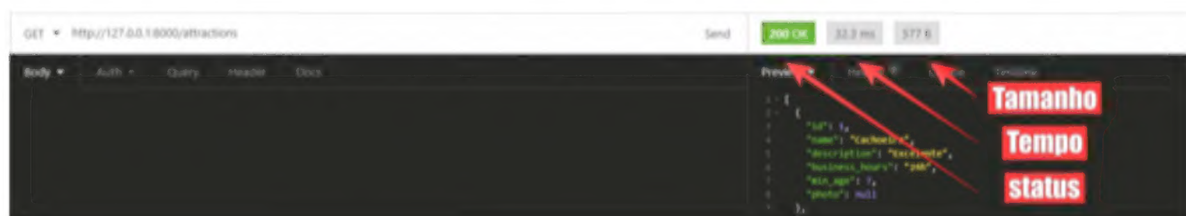
Neste exemplo irá mostrar as atrações turísticas através da url http://localhost:8080/atracoes, portanto, ficará da seguinte forma:



Após você adicionar um endereço de uma API válido, indicando para fazer a requisição, basta apertar o botão *send*. Feito isso a resposta da API irá aparecer à direita da tela do insomnia:



É importante ter a percepção que iremos verificar a apresentação do status da resposta, o tempo decorrido e seu tamanho.



Sabemos que esse tipo de informação se faz muito relevante para podermos certificar que sua API está realmente retornando os códigos que foram recomendados na arquitetura REST, além claro de servir para verificar como foi o desempenho da sua API.

Agora que você já viu como se cria uma requisição GET, pedimos que implemente também as requisições PUT, POST e DELETE.



Gabarito Atividade Prática

Para responder esta questão é necessário primeiramente criar uma API REST ou mesmo utilizar alguns modelos de APIs públicas que tem disponíveis na internet e que também indicamos no texto. Feito isso, irá seguir um passo a passo para fazer os testes de consumo da API através do insomnia, executando as requisições GET, PUT, POST e DELETE.

Agora vamos usar o Json server para simular uma API de cadastro de livros.

Mais informações sobre Json server pode ser encontrada na URL abaixo:

https://github.com/typicode/json-server

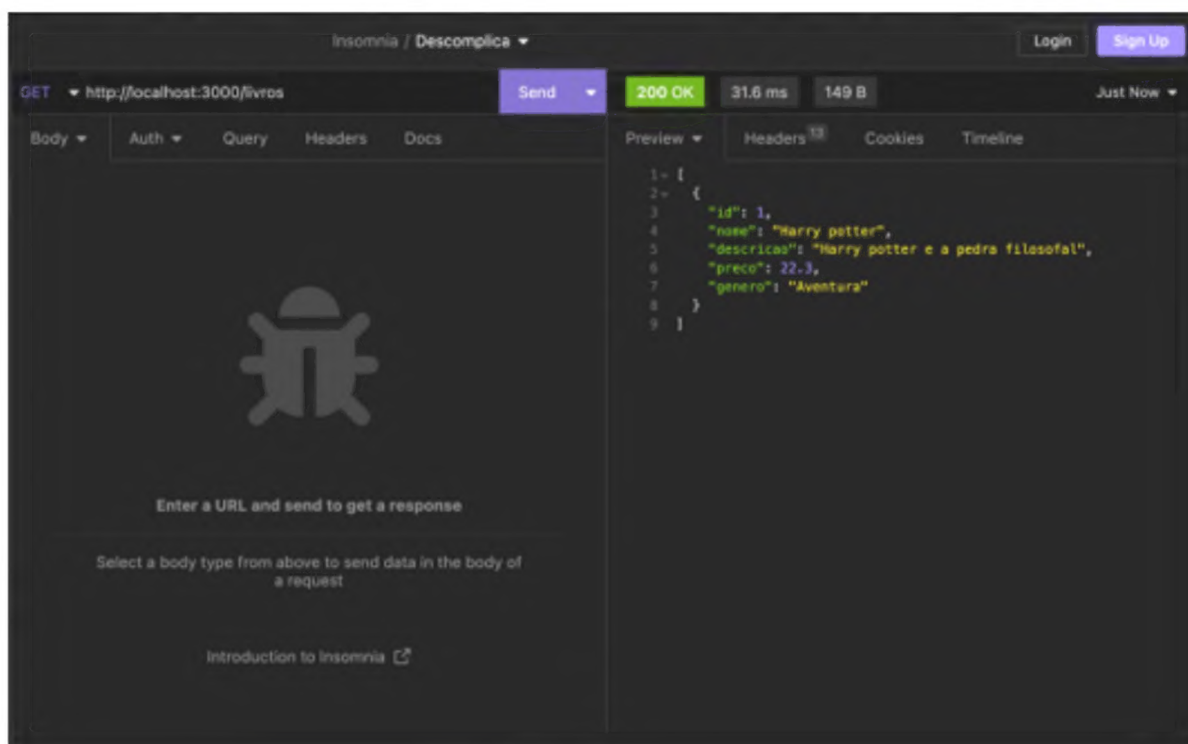
Primeiro vamos criar o objeto Json

```
{
  "livros": [
    {
      "id": 1,
      "nome": "Harry potter",
      "descricao": "Harry potter e a pedra filosofal",
```

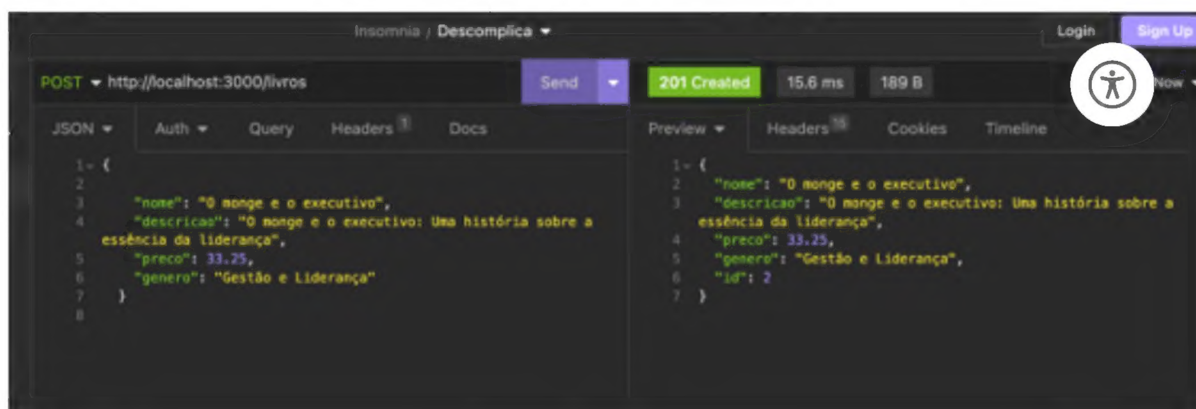


```
        "preco": 22.30,
        "genero": "Aventura"
      }
    ]
}
```

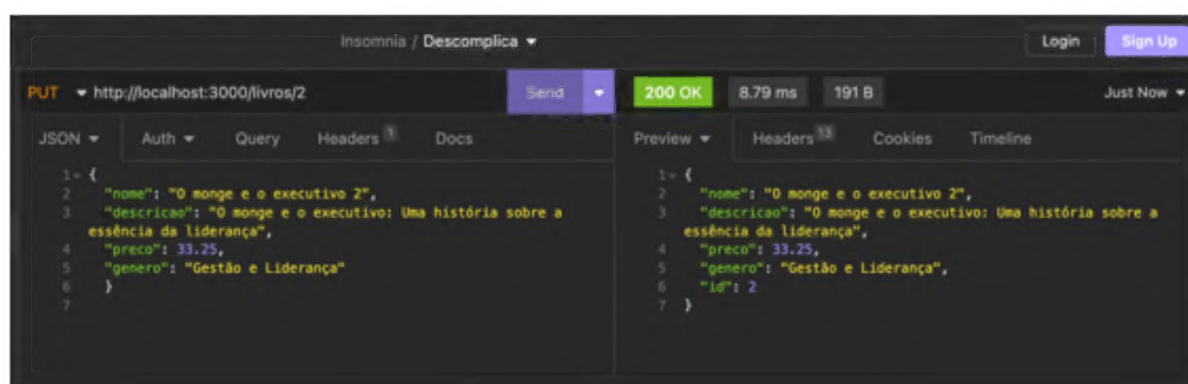
Agora vamos criar a conexão HTTP GET no insomnia



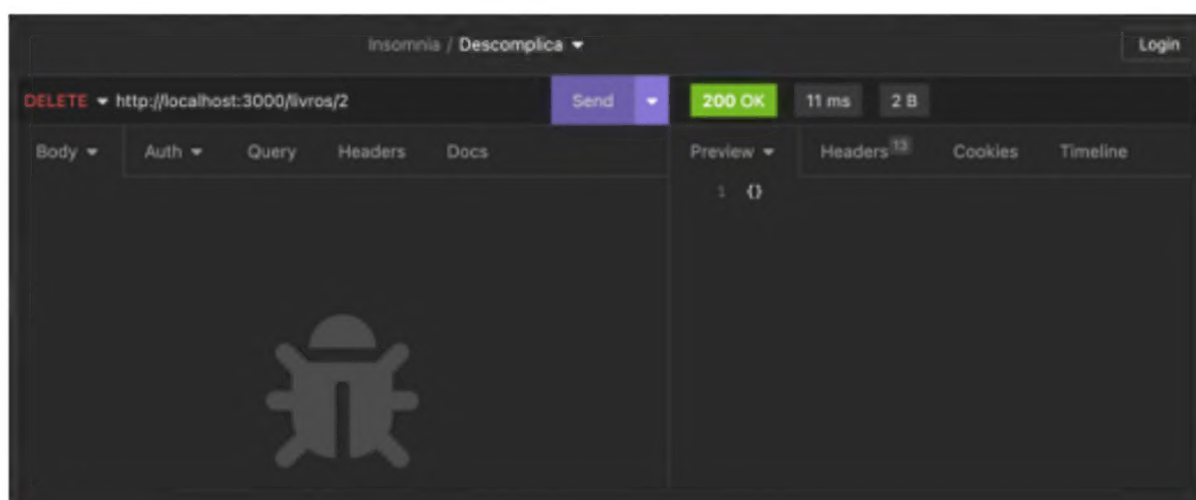
Agora vamos criar uma conexão HTTP POST



Agora vamos criar uma conexão HTTP PUT



Agora vamos criar uma conexão HTTP DELETE



Ir para exercício